SESSION DEPENDENT DE-IDENTIFICATION OF ELECTRONIC MEDICAL RECORDS

A Thesis

Presented in Partial Fulfillment of the Requirements for

the Degree Bachelor of Science with Honors Research Distinction in

Electrical and Computer Engineering

By

Stephen Kahmann

* * * * *

Electrical and Computer Engineering

The Ohio State University

2012

Examination Committee:

Prof. Bradley D. Clymer, Adviser

Prof. Barbaros Selnur Erdal

ABSTRACT

The decreasing cost of both computing power and storage has lead to the widespread adoption of Electronic Medical Records in modern health care facilities. This movement has created a valuable source of diagnostic and patient information for medical research. While these data could provide valuable insight in the study of disease, patient privacy is of utmost concern and the protected patient information must be de-identified. This de-identification process can introduce error into the data-set.

This thesis implements de-identification techniques developed by The Ohio State University Medical Center Information Warehouse to de-identify electronic medical records, and analyzes the effect the de-identification of patient visit dates has on the reliability of the resulting data-sets. In addition, we develop a tool that allows researchers to characterize the reliability of de-identified data-sets based on patient population counts.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

The decreasing cost of both computing power and storage has lead to the widespread adoption of Electronic Medical Records (EMRs) in modern health care facilities. This technology has allowed medical professionals to more efficiently and accurately store patient's personal and diagnostic information. The proliferation of this technology has, however, created another added benefit to the health care field - the use of EMR databases in medical research. With powerful databasing and querying tools, researchers can now analyze millions of patient demographic and diagnostic records to aid in the discovery and combat of disease.

A major concern in the use of EMRs for medical research is the assurance of patient privacy. In the United States, the Health Insurance Portability and Accountability Act (HIPAA) of 1996 protects the rights of human subjects used for research by declaring certain aspects of patient records to be Protected Health Information (PHI).

Motivation for this project comes from existing de-identification techniques developed by The Ohio State University Medical Center Information Warehouse (OSUIW). These techniques use Oracle SQL scripts to modify patient identification numbers and

medical record dates in order to de-identify certain aspects of EMRs to be used for diagnostic research.

The process of de-identifying patient information by modifying medical record dates can have an impact on the accuracy of the data being analyzed. For instance, if a researcher were interested in examining the number of patients that suffer from lung cancer in a given month of the year, the act of de-identifying dates may lead to a discrepancy in the de-identified count versus the real data. This discrepancy can cause error in a researcher's analysis.

This project focuses on the design of a tool that incorporates the OSUIW de-identification techniques to analyze the reliability of de-identified electronic medical records. It gives researchers an estimation of the required size of a patient population in order to bound the error by a specified amount.

1.2 Standards and Regulations

The use of electronic medical records for research brings up many concerns about the privacy of patient information. This project refers to two mandates regarding patient privacy: the Health Insurance Portability and Accountability Act (HIPAA) of 1996 and the "Honest Broker" status granted to the OSUIW by The Ohio State University.

1.2.1 Health Insurance Portability and Accountability Act

One of the major goals of the Health Insurance Portability and Accountability Act of 1996 is to assure protection of patients' health information without impeding the sharing of health information needed to provide high quality health care. The privacy rule portion of the act establishes specific data items as "Protected Health

Information (PHI)". These data items include many common identifiers (e.g., names, addresses, dates, telephone numbers, email addresses, and Social Security Numbers).

The key point in this act is that there are no restrictions on the use or disclosure of de-identified health information. De-identified information neither identifies nor provides a reasonable basis to identify an individual. This can be done through the removal or modification of specific identifiers of the individual so that the resulting information cannot be used to identify the individual. [1][2]

1.2.2 OSUIW Honest Broker Status

The Internal Review Board (IRB) at The Ohio State University Medical Center has granted the OSUIW the "Honest Broker" status, which allows IW analysts to provide de-identified clinical data for research. The stipulations provided by the IRB are that the patient cannot be re-identified by the data and that the data set must contain more than 25 patients.

1.3 De-identification Methods

As discussed in section 1.2.1, specific identifiers must be removed or modified in order to de-identify the patient data so it can be used for research. In this study we focus on de-identifying medical record numbers (MRN) and dates such as patient admit and discharge dates. The de-identification process takes place on a session dependent basis so each new connection to the database results in a unique set of de-identified data.

The de-identification of the MRNs consists of a session dependent random consistent offset to all the MRNs. For instance, 1000 would be added to every MRN. The dates on the other hand are de-identified with a random, patient consistent date shift

within a specific window size. For example, if the shift window size is two weeks, then every patient is randomly assigned a shift amount +/-7 days and all of their admit and discharge dates would be shifted by that amount.

1.4 Teamwork Component and Acknowledgments

This project was conducted in collaboration with the OSUIW and the research group led by Prof. Bradley D. Clymer. The contact at the OSUIW was Prof. Barbaros Selnur Erdal who supplied the patient data, de-identification algorithms, and expertise in the area of Oracle Databases and de-identification methods. Prof. Clymer's research group provided valuable feedback at many stages of the project.

This project is built upon previous work completed by the OSUIW and was carried out with the intention of being made available for and integrated with other research endeavors at The Ohio State University.

1.5 Problem Statement

Electronic medical records offer a vast source of data for medical research. Standards and regulations restrict the use of patient data for research unless the patients are un-identifiable. The process of de-identifying this data introduces error into the result set.

This project takes sample medical records and de-identifies them using techniques developed by the OSUIW. Tests were run on the data in order to develop analysis techniques to determine the effects modification of dates has on the accuracy of patient population data. A tool was then developed to allow researchers to determine the accuracy of their data tests based on our analysis techniques.

1.6 Organization

This document is divided into 4 additional chapters. Chapter 2 discusses the database organization. This chapter details the data and database schema and gives a technical explanation of the de-identification process. Chapter 3 discusses the data analysis portion of the project including the tests being run as well as the tools and technology used to run the tests and analyze the test data. Chapter 4 discusses the tool being designed to allow users to run tests against a de-identified database and view the statistical results. Finally, chapter 5 details the future uses for this project and what improvements can be made to increase its functionality and accuracy.

CHAPTER 2

DATABASE ORGANIZATION

2.1 Database Introduction

The data used in the project was provided by the Ohio State Medical Center Information Warehouse (OSUIW). It consists of patient records previously de-identified using their de-identification techniques. These patient data were loaded into an Oracle Enterprise Database running Oracle version 11g. After the data were imported into the database, the SQL scripts developed by the OSUIW were integrated into the environment making it ready to serve de-identified data to the testing clients developed for this project.

This chapter describes the data and database schema used in our Oracle database, and gives a technical discussion on the de-identification methods used including algorithms and performance considerations.

2.2 Data and Schema

The OSUIW provided us with two sets of data for this project. The first set contains roughly 5.7 million patient records of almost 1 million distinct patients. The second, larger set contains about 20 million records, again with roughly 1 million

distinct patients. The difference between these two data-sets is how the diagnosis codes are handled. In the smaller set, each record describes one patient visit with a primary diagnosis code. The larger set, on the other hand, consists of one record for each diagnosis code for each visit of every patient. This includes a column of diagnosis code type (admit or discharge) and diagnosis code rank. Both data sets contain columns for admit and discharge dates, medical record numbers (MRN), and diagnosis codes. An example of the database setup for the smaller set of data is shown in Table 2.1, and for the larger set in Table 2.2

DSCH_DT	ADM_DT	MRN	PRIM_DXCD
11-NOV-97	11-NOV-97	83345064929	174.9
21-OCT-97	21-OCT-97	83345064929	174.8
21-OCT-97	21-OCT-97	85225710612	578.9

Table 2.1: An example showing the table setup for the smaller set of patient data.

DSCH_DT	ADM_DT	MRN	DX_CD	DX_TYP	DX_RANK
13-JAN-96	13-JAN-96	88331573923	038.42	DISCHARGE	3
13-JAN-96	13-JAN-96	88331573923	112.4	DISCHARGE	2
13-JAN-96	13-JAN-96	88331573923	150.9	DISCHARGE	1
11-MAY-96	11-MAY-96	84439184551	414.9	DISCHARGE	2
11-MAY-96	11-MAY-96	84439184551	V45.81	DISCHARGE	1

Table 2.2: An example showing the table setup for the larger set of patient data.

The main focus of this project was the modification of visit dates for de-identification, so the additional diagnosis code data in the larger data-set was not needed. Therefore, in order to decrease disk usage and query times, only the smaller data set was

used. If a future project were to focus on diagnosis codes, for example in looking at re-identification risk, the larger data set may prove to be more useful.

The database schema used for this project was very simple. One table was created for each data-set. The date columns used the 'DD-MON-YY' format, the MRNs were NUMBERS, and the diagnosis code columns used VARCHAR2 types. One important aspect was that it was necessary to use the 'DD-MON-RR' format when importing the dates into a table. This tells the database that the 9X dates belong in the 20th century and the 00 and greater dates belonged in the 21st century.

Indexing in tables of this size is very important. For each table we built bitmap indexes on both the date and diagnosis code columns. Bitmap indexes provide a powerful method for quickly retrieving results for difficult queries and dramatically decreased our query times.

2.3 De-identification Methods

The de-identification process for this project focused on de-identifying medical record numbers (MRN) and patient visit dates. MRNs were de-identified by adding a random, consistent offset to each patient ID. For example, each MRN would be increased by 1000. The visit dates were de-identified by applying a random offset to each date that was consistent for each patient. That is, every patient was randomly assigned a shift amount within a specified window (e.g. +/-7 days), and every date for that patient was shifted by that amount.

The graphic in Figure 2.1 shows an example of this de-identification process. The top portion of the text displays the original patient information, while the bottom portion displays the data after de-identification. From the figure you can see that

the MRNs have been consistently offset and the dates have been randomly modified. The highlighted rows show how the dates for each patient are shifted consistently.

MRN	Admit Date	Discharge Date	Dx Code
86224730409	16-Dec-96	16-Dec-96	429.9
83363196412	24-Dec-96	24-Dec-96	611.9
87851815591	06-Nov-96	06-Nov-96	729.5
82912230151	07-Dec-96	07-Dec-96	786.50
82912230151	04-Jul-95	06-Jul-95	V76.12

MRN	Admit Date	Discharge Date	Dx Code
86224731302 (893)	16-Dec-96 (0)	15-Dec-96 (-1)	429.9
83363197305 (893)	24-Dec-96 (0)	23-Dec-96 (-1)	611.9
87851816484 (893)	06-Nov-96 (0)	03-Nov-96 (-3)	729.5
82912231044 (893)	07-Dec-96 (0)	11-Dec-96 (+4)	786.50
82912231044 (893)	04-Jul-95 (0)	10-Jul-95 (+4)	V76.12

Figure 2.1: Graphic depicting the two de-identification methods used in this project.

2.3.1 De-identification Algorithms

The SQL scripts controlling the de-identification of the patient data consisted of two parts. The first element was a set of SQL functions and procedures that provided the random numbers used to modify the data. The second element was a SQL logon trigger that initialized the random number seeds for these functions and procedures. This setup created a session dependent de-identification environment that provided a unique set of de-identified patient data for every login to the database.

The initialization for the random number seeds were controlled by two procedures: INIT_MRN and INIT_DT. Both of these procedures were called by the logon trigger on connection to the database. Each procedure simply made a call to the built-in Oracle function DBMS_RANDOM.VALUE(low NUMBER, high NUMBER) RETURN

NUMBER. Given a NUMBER low and a NUMBER high, this function returns a random NUMBER that is greater than or equal to low and less than high.

The MRNs were de-identified by adding a consistent random offset to each patient ID. This was accomplished by calling the function GET_MRN() which simply returned the NUMBER generated in the initialization step by INIT_MRN().

The visit dates were handled in a different way in order to de-identify them randomly on a patient consistent level within a shift window. A procedure was created, SET_DATE_OFFSET(offset NUMBER), which allowed the user to set the shift window to be +/- offset. After the date offset was fixed, the user would call RAN_DT(mrn NUMBER) in order to retrieve the date offset to be applied to the patient visit date. This function returns a random number between -offset and +offset based on the MRN number of the patient. The equation detailed in Equation 2.1 describes how RAN_DT(mrn NUMBER) determines the date shift for each patient.

$$\begin{aligned} \text{R\_DT} := (2 * mod(mod(\text{mrn} + \text{GET\_DT}(), 7), 2) - 1) * \\ (mod(\text{mrn} + \text{GET\_DT}(), \text{GET\_DATE\_OFFSET}()) + 1) \end{aligned} \tag{2.1}$$

2.4 Accessing De-identified Data

Given that the de-identification algorithms allow the user to simply call functions to modify MRNs and dates, there exist many ways for the user to create and access de-identified data. This section discusses two methods of accessing de-identified patient data. The first method, the SQL View Approach, takes advantage of the dynamic nature of SQL views to de-identify data on-the-fly. The second method uses the de-identification algorithms to create a separate table in the database that contains de-identified data. This method is called the De-identified Table Approach.

After investigating and testing these two approaches, the De-identified Table Approach proved to be more efficient and was chosen as our primary de-identification method due to performance considerations. A detailed explanation of each approach is included for possible future refinements.

2.4.1 SQL View Approach

The SQL view approach consists of creating a SQL view that sits on top of the original patient data. When a user sends a query to the view, the data are de-identified at query time. The graphic in Figure 2.2 depicts the view's SQL code and shows how this method takes advantage of the de-identification algorithms described in section 2.3.1.

```
Select
  A.P_KEY,
  A.MRN + DEIDVW.GET_MRN() MRN,
  A.DSCH_DT + DEIDVW.RAN_DT(A.MRN) DSCH_DT,
  A.ADM_DT,
  A.PRIM_DSCH_DX_CD
FROM
  SMALL_PATIENT_DATA A
```

Figure 2.2: The SQL Code within the view taking advantage of the de-identification algorithms

There are a couple advantages offered by the View Approach resulting from SQL views being in-memory only. This dynamic nature ensures session dependence and efficient use of disk space. In addition, use of the views adds a layer of security by hiding the original patient data from the user.

The problem surrounding the use of SQL views is their ability to push predicates, such as dates and diagnosis codes, to the table for searching and sorting with the

table's indexes. Because the columns for dates are de-identified, the database management system (DBMS) is unable to apply indexes to the query. This results in poor performance times. The time taken to run a simple count query against the patient data table with indexes is roughly 0.5 seconds while the query run against the SQL view takes around 8 minutes. This poor performance led us to choose the De-identified Table Approach as our primary de-identification method.

2.4.2 De-identified Table Approach

The De-identified Table Approach consists of running a script to create a separate table containing de-identified versions of every record in the original patient table. After this table is created, indexes are built on the columns and a user can simply query that table as if it were the original data. This results in very fast performance times of about 0.5 seconds per query.

The disadvantages to this approach are the overhead required to create the de-identified data table, and the disk space required to hold it. While these are a major problem in enterprise settings, this method works very well for our testing as we plan to run thousands of queries against the database to collect our statistics for analysis.

The graphic in Figure 2.3 shows the script used to create the de-identified table and its indexes. From the figure you can see how this method takes advantage of the de-identification algorithms described in section 2.3.1 and the bitmap indexes created on the date and diagnosis code columns.

This chapter discussed the the data and database schema used in our Oracle database. It provided technical explanations of our de-identification methods, and

```
INSERT INTO ENC_DEID SELECT
  A.DSCH_DT + DEIDVW.RAN_DT(A.MRN) DSCH_DT,
  A.ADM_DT,
  A.MRN + DEIDVW.GET_MRN() MRN,
  A.PRIM_DSCH_DX_CD
FROM SMALL_PATIENT_DATA A;

CREATE BITMAP INDEX ENC_DEID_DSCH_DT ON ENC_DEID (DSCH_DT ASC);
CREATE BITMAP INDEX ENC_DEID_DXCD ON ENC_DEID (PRIM_DSCH_DX_CD ASC);
CREATE BITMAP INDEX ENC_DEID_DT_DXCD ON ENC_DEID (DSCH_DT ASC,
     PRIM_DSCH_DX_CD ASC);
```

Figure 2.3: Graphic detailing the script used to create the de-identified table and indexes.

described two methods of accessing that de-identified data. The next chapter discusses the tests we ran against the de-identified data and how we analyzed the results.

CHAPTER 3

TESTING AND DATA ANALYSIS

3.1 Data Analysis Introduction

The purpose of this project was to analyze the error introduced into a data-set caused by modifying dates in patient records. In order to analyze this error we de-identified patient data in an Oracle database using methods described in chapter 2. Next, using the Java programming language we wrote code to execute queries against these patient data and to collect results about patient population data. After these data were collected we wrote functions in MATLAB to analyze the test data and create visualizations of the results.

This chapter discusses the test setup in the Oracle database and how we implemented the tests using Java. Additionally, it describes the analysis and visualization of the test data using MATLAB.

3.2 Test Setup and Implementation

The purpose of the testing in this project was to collect data on how the de-identification of medical records introduced error into the resulting data-set. To accomplish this task patient data were de-identified in an Oracle database, and Java was used to run many queries against these data to collect samples of patient populations.

3.2.1 Testing Method

The basic test consisted of counting the original number of patients in a particular time window with a specific diagnosis code followed by counting the number of patients in that same time window after de-identifying the patient records. There could be differences in that count due to some patients being shifted out of the test window and a different number patients being shifted into the test window. This process was repeated numerous times with various time windows and date shift amounts in order to look at possible trends in error related to population sizes and the ratio of the maximum number of days shifted to the number of days in the time window (e.g. a ratio of 0.5 being a maximum of +/- 14 day shifts in a 28 day time window). Many samples were taken of each test in order to get an average RMS error for each date shift amount and time window size.

Test Data

The tests for this project were conducted on data from a five year time period from 1999-2003. Five different time windows were chosen to be tested with five day shift-to-time window ratios for each time window under test. This resulted in 25 tests run against the database. The time windows selected for testing were 2 weeks, 1

month, 2 months, 6 months, and 1 year. The day shift-to-time window ratios chosen were 0.1, 0.2, 0.3, 0.4, and 0.5.

An example of these tests would be the five tests run for the one month (28 day) time window. This test set consisted of five tests to test each ratio against that time window. The five tests for this window were:

- 28 day window (+/-) 2 day shift
- 28 day window (+/-) 5 day shift
- 28 day window (+/-) 8 day shift
- 28 day window (+/-) 11 day shift
- 28 day window (+/-) 14 day shift

Each 28 day test window contained 65 samples, one for each 28 day period over the 5 years under test, and provided one of the twenty-five error data points.

Test Performance

The performance of the tests depends on two factors: the number of times de-identification needed to be performed (the number of tests being run) and the number of diagnosis codes beings tested. The equation to estimate the time required to run the tests is show in equation 3.1.

$$(deid\ time * \#\ tests) + (\#\ queries * query\ time * 2 * \#\ dxcds) \tag{3.1}$$

In our testing the de-identification time (deid time) took approximately 2 minutes, we were running 25 tests, the number of queries run was 242 (five time window samples over 5 years), query time was about 0.5 seconds, and the number of diagnosis codes (dxcds) was determined by the user. Tests running with one diagnosis code took

about one hour to complete with each additional diagnosis code adding about 10 minutes to the test. While these numbers offer a good estimate to test duration, performance varies according to server setup and database load.

3.2.2 Querying De-identified Table

Querying the database to retrieve data takes two steps: setting the maximum date shift amount and counting the number of patients with a specific diagnosis code in a particular time window. Setting the maximum date shift amount is explained in section 3.2.3, while the query to count the number of patients in the desired window in show is figure 3.1.

```
SELECT COUNT(*) FROM <table>
WHERE
PRIM_DSCH_DX_CD = '<dxcd>'
AND
TO_CHAR(DSCH_DT, 'DD-MON-RR') >= TO_DATE('<startDate>', 'DD-MON-YYYY')
AND
TO_CHAR(DSCH_DT, 'DD-MON-RR') < TO_DATE('<endDate>', 'DD-MON-YYYY');
```

Figure 3.1: Graphic depicting the query to count the number of patients in a particular time window.

The highlighted text shows the input parameters to the query. The 'table' parameter determines which table is being queried (original or de-identified data), the 'dxcd' parameter describes which diagnosis code is being used to filter the patients, and the 'startDate' and 'endDate' parameters determine the time window under test.

3.2.3 Java Testing Tools

The Java code to execute the tests is very straight-forward. The code simply connects to the database using Java JDBC, sends the proper SQL queries according

to the tests to be performed, and saves the data in the correct file format as described in section 3.3.1. One important task for the Java program is to set the maximum day shift for the de-identification algorithms. This is done by executing the following query: BEGIN DEIDVW.SET_DATE_OFFSET(?); END;

3.3 Test Data Analysis

The goal of the test analysis was to find the RMS error for the twenty-five tests described in the previous section and create a way to visualize these results to discover trends. MATLAB was used to take in the data collected from the Java program, calculate the RMS error, and create an interpolated surface plot of the error.

3.3.1 Input Parameters

The MATLAB program requires a particular file format for the test data. Each test set must be contained in its own file according the following naming convention: 'w<*window size*>_s<*shift amount*>_list.txt', for example, 'w14_s7_list.txt'. Each file contains all the samples for every diagnosis code tested for that test following the file format example in table 3.1.

1	74	V76.12	78	79	-1.28
2	74	V76.12	125	124	0.80
3	74	V76.12	136	131	3.68
...					
131	74	174.9	443	452	-2.03
132	74	174.9	333	325	2.40
133	74	174.9	358	332	7.26

Table 3.1: An example showing the file format for the test analysis input data.

This example shows that the file consists of columns for the test number, session ID, diagnosis code, original population count, de-identified population count, and test error percentage all separated by tabs.

3.3.2 Analysis Algorithm

The algorithm used to determine the RMS error is shown in equation 3.2. The equation shows that we found the RMS for all the samples from each test (e.g. the 65 samples described in section 3.2.1) and divided that by the average patient count across those samples to get the RMS error.

$$error = \sqrt{\frac{\sum [(org.\ \ count - deid\ \ count)^2]}{\#\ \ of\ \ samples}} / (mean\ \ original\ \ count) \tag{3.2}$$

This calculation was performed on all twenty-five test sets resulting in twenty-five error points that each corresponded to a population size (test window) and a shift-to-window ratio.

3.3.3 Analysis Output

After the twenty-five RMS error points were calculated, we needed to plot the data with an interpolated surface in MATLAB. Not only did we want to create a function that would plot the error so we could visualize the trends for the diagnosis codes, we also wanted to be able to save the interpolated function. This would allow us to call it at a later time to determine the interpolated error at data points not collected during testing. To achieve these goals we decided to the the `TriScatteredInterp(X, Y, V)` function.

The `TriScatteredInterp(X, Y, V)` function creates an interpolant function that fits a surface of the form V = F(X, Y) to the scattered data in (X, Y, V). One

important thing to note about this function is that it only returns values within the original bounds of interpolation. Any values of X or Y outside of this boundary simply returns NaN for not a number. More information on this function can be found in the MathWorks product documentation.

Figures 3.2 and 3.3 show two examples of the RMS error parameterization for two diagnosis codes tested. In the plots the two horizontal axes describe the population count vs. the shift-to-window ratio, and the vertical axis describes the RMS error. As one can see from the two plots, each diagnosis code shows unique characteristics in how the error changes according to the population count and shift-to-window ratio. The first displays an almost constant error as population count increases with a somewhat linearly increasing error as the shift-to-window ratio increases. The second, however, shows an almost exponential decrease in error as the population increases, while remaining relatively constant in error as the shift-to-window ratio increases.

This chapter discussed the test setup and implementation for our project. It then detailed our analysis steps and described our visualization methods. The next chapter discusses the testing tool developed to help researchers determine the reliability of de-identified patient data.

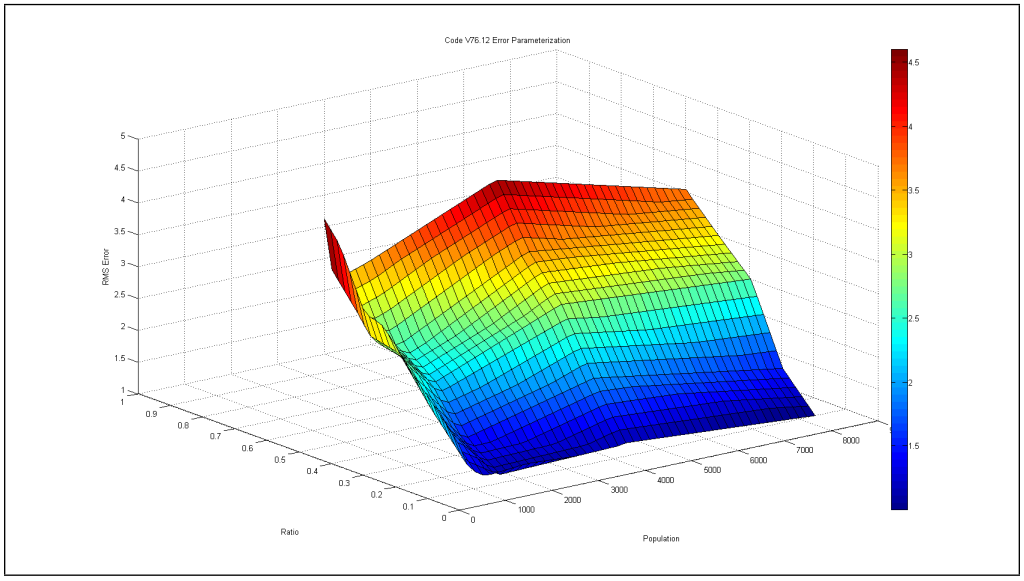

Figure 3.2: Graphic depicting the interpolated surface formed from the test data for a test of one diagnosis code.

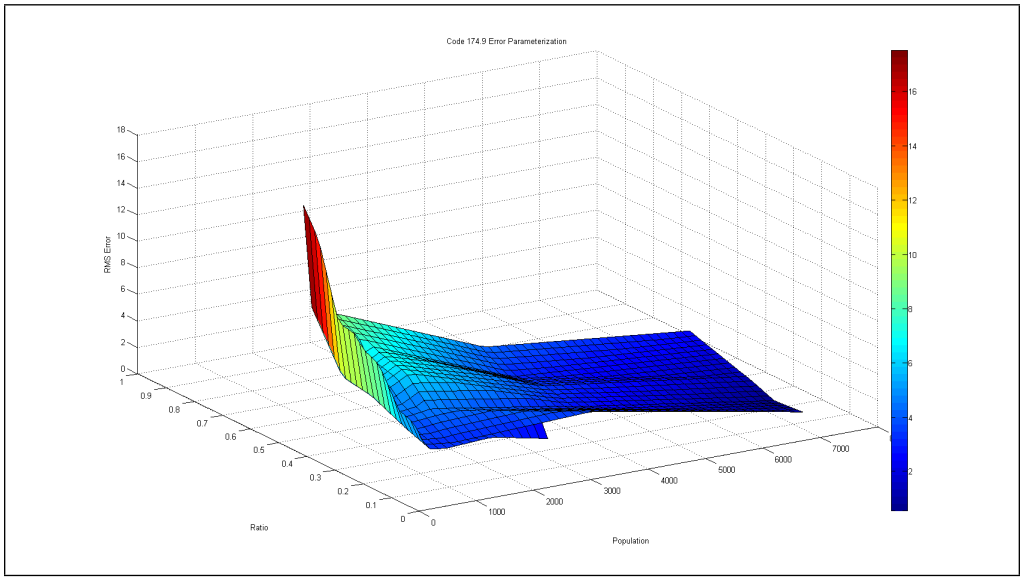

Figure 3.3: Graphic depicting the interpolated surface formed from the test data for a test of another diagnosis code.

CHAPTER 4

DATA TESTING TOOL

4.1 Testing Tool Introduction

The purpose of the testing tool is to give researchers a tool that will allow them to determine the minimum patient population needed in order to reliably use de-identified data. For example, if a user wanted to study how many patients suffered from lung cancer in a particular month of the year, this tool would tell them how reliable the de-identified data would be, given the rate of occurrence for lung cancer in a given month. This tool was developed using Java with the integration of the MATLAB analysis tools discussed in chapter 3

This chapter discusses the steps taken to design this tool and how we integrated MATLAB into the Java program.

4.2 Tool Design

The design of this tool went through various steps in order to determine the program's requirements, input parameters, program flow, and output to the user. Throughout the design process, iterations were presented to my adviser Dr. Clymer and he provided feedback on the interface and operation of the program that shaped its development.

After requirements analysis, it was clear that this program needs to offer a clean, simple interface to allow a researcher to easily enter diagnosis codes for test. It must then provide output that is easy to understand and interpret. We settled on a final design that prompted the user to enter diagnosis codes and start the test. The program then runs the tests while displaying the progress to the user. Once the tests are completed, the tool then outputs three common error percentages (5%, 2%, and 1%) and the associated minimum patient population that would ensure this reliability according to our analysis. In addition to these statistics, the tool also displays the interpolated RMS error surfaces so the user can analyze the trends associated with each diagnosis code.

An additional element of the design was to perform caching of results for previously run diagnosis codes. This process can save a significant amount of time in returning results to a user. As previously mentioned in section 3.2.1, analysis of one diagnosis code can take up to one hour, while taking advantage of cached analysis results can return results in seconds.

4.3 Tools and Technology

The Java programming language along with the tool MATLAB Builder JA were used to develop this testing tool. MATLAB Builder JA is a tool provided by MathWorks to wrap MATLAB functions into Java components for use when developing Java applications. Figure 4.1 shows an example of the MATLAB Builder JA interface and how it packages MATLAB functions into a Java class.

This Java class allows the Java program to make calls to MATLAB functions complete with input parameters and return values. An example of this idea is shown

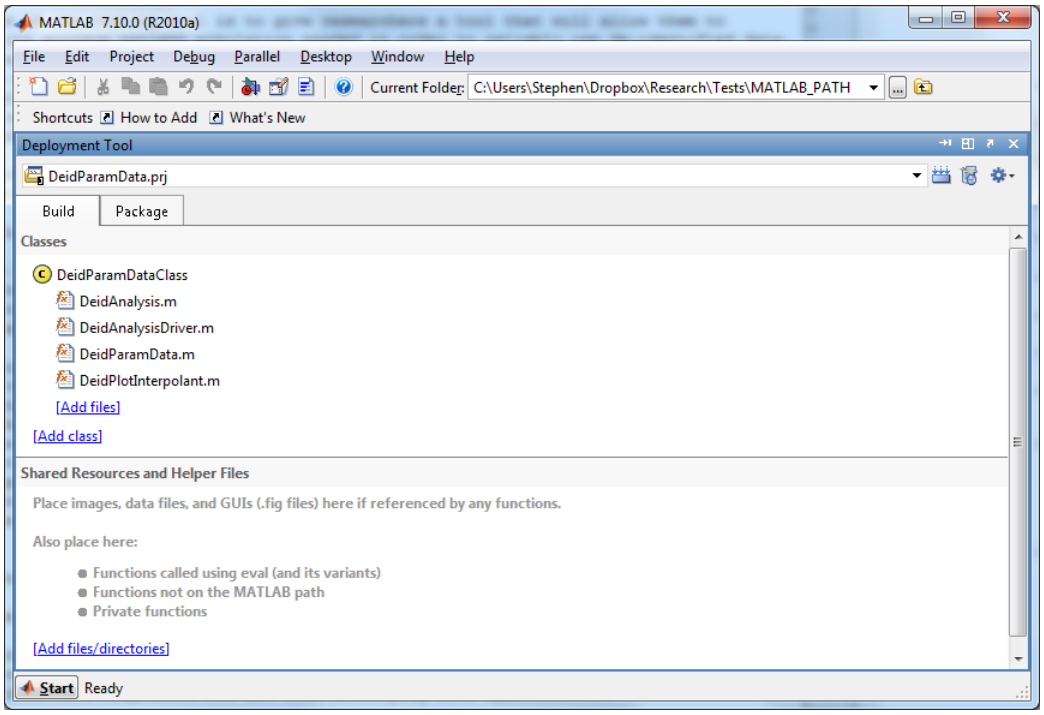

Figure 4.1: Screenshot of the MATLAB Builder JA interface and packaging MATLAB files into a Java class.

in Figure 4.2. This example shows how to call a MATLAB function from the generated Java class and convert the output to useful Java components.

4.4 Testing Tool

The final form of the testing tool consists of two views. The initial view simply provides instructions, a text field for entering diagnosis codes, and a button to begin the test. A screenshot of this view can be seen in Figure 4.3. The analysis view takes precedence once the button is pressed to begin the test. This view provides feedback on the progress of the test followed by the test results once the tests are finished. A screenshot of this view can be seen in Figure 4.4.

```
Object[] result = null;
result = matlabComp.DeidAnalysisDriver(1, dxcd, error);

MWNumericArray numResult = (MWNumericArray) result[0];
double[] doubleResult = numResult.getDoubleData();
```

Figure 4.2: Graphic depicting the call to a MATLAB function encapsulated within a Java class.

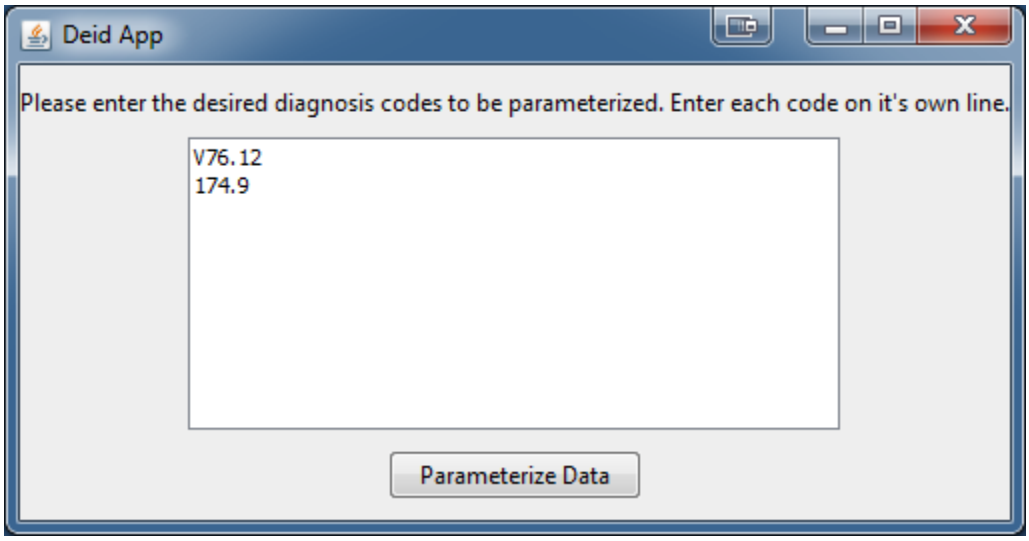

Figure 4.3: Screenshot of the initial view of the testing tool.

In the analysis view of Figure 4.4 one can see the table describing the minimum patient populations required to ensure a specific amount of reliability and the interpolated surface from the MATLAB functions.

Once the testing tool was complete, it was presented to Prof. Barbaros Selnur Erdal, our contact with the OSUIW, for final verification.

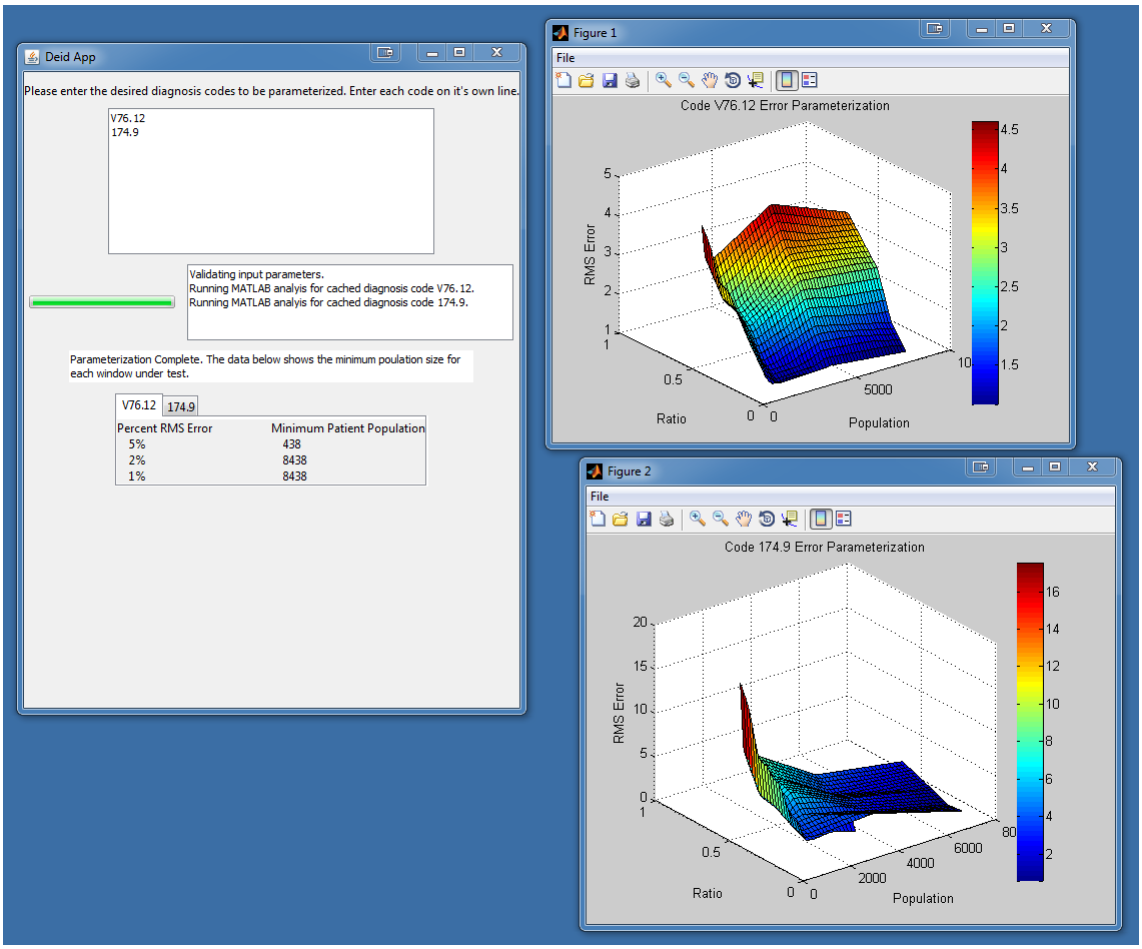

Figure 4.4: Screenshot of the analysis view of the testing tool.

CHAPTER 5

SUMMARY AND FUTURE WORK

5.1 Summary

This document discussed the database organization for the project, detailing the data and database schema, and gave a technical explanation of the de-identification process. Next, it described the data analysis portion of the project including the tests being run as well as the tools and technology used to run the tests and analyze the test data. Finally, it discussed the tool designed to allow users to run tests against a de-identified database and view the statistical results.

5.2 Future Improvements and Uses

While this project takes steps toward making de-identified electronic medical records a reliable source of medical research information, there are some improvements that could be made in the future. For one, the interpolated function created by MATLAB can only return interpolated error values, so any test situation that lies outside those boundaries cannot be tested using the tools discussed here. A possible solution to this problem could be to fit exponential curves to the test data in order to extrapolate past the test boundaries

Another future improvement to the project could be to the testing tool interface. To make the tool more accessible to researchers, the tool could be encapsulated in a web servlet to allow users to connect to and use the tool from anywhere.

BIBLIOGRAPHY

[1] Health insurance portability and accountability act of 1996, August 1996. Public Law 104 - 191.

[2] U.S Department of Health and Human Services. Summary of the hipaa privacy rule, May 2003.